DOCUMENT RESUME

ED 250 720 CS 208 694

AUTHOR Skulicz, Matthew
TITLE Some Analogies between Computer Programming and the Composing Process.
PUB DATE Oct 84
NOTE 28p.; Paper presented at the Annual Meeting of the New York State English Council (34th, Amherst, NY, October 18-19, 1984).
PUB TYPE Speeches/Conference Papers (150) -- Viewpoints (120) -- Guides - Classroom Use - Guides (For Teachers) (052)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Cognitive Processes; Coherence; *Computer Literacy; *Computer Software; Editing; Essays; Higher Education; *Programing; Student Attitudes; Teaching Methods; *Writing (Composition); *Writing Processes
IDENTIFIERS *Audience Awareness

ABSTRACT

Since there are similarities between the process of writing computer programs and the process of writing successful expository prose, a student's knowledge of computer programing can contribute to the understanding of some principles of composition. The establishment of a clear objective is the first priority of both the writer and the programer, who must then separate the problem into its parts and discover the relationship between these parts. Then the individual steps necessary to realize the various parts of the outline or flowchart must be constructed. In the cases of both writing and programing, the constraints imposed by the audience--the reader or the computer--on the finished product are the same (although different in degree). Thus, because computers demand explicit step-by-step instructions, computer programers are forced into deliberate planning, clear and coherent thought, and explicit expression. Such valuable habits are easily transferable to the composing process. A computer program may also be viewed as a system of parts that solve a problem. An essay is a similar problem solving system, with the problem being the elucidation of the thesis statement. If students can be shown that an essay, like a program, is a system, they then have at their disposal a set of attitudes and concepts for understanding the composing process. (HOD)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
X This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Matthew Skulicz

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

ED250720

SOME ANALOGIES BETWEEN COMPUTER PROGRAMMING AND THE COMPOSING PROCESS

by Matthew Skulicz
Erie Community College/City
Buffalo, New York 14203

Recently, a great deal of work has been done, and continues to be done, on accommodating computers to English instruction. In general, this work takes two directions. Either it explores the use of word-processing programs by writing students, or else it attempts to evaluate the recent software dealing directly with the composing process, particularly in the areas of invention and organization.[1]

My purpose here is to take a different tack. I do not propose to discuss applications of computers in the English classroom. Rather, I wish to look from an English perspective at computer literacy, that is, the student's knowledge of computers and ability to use them. I would like to focus, in particular, on the student's ability to write programs for the computer, and to discuss some similarities between the process of writing computer programs and the process of writing successful expository prose. I

CS 208 694

would like to speculate on how a student's knowledge of computer programming can contribute to his or her understanding of some of the principles of composition. I will contend that students who know how to program computers have developed a set of concepts which they can use to clarify their understandings of the composing process.

To begin, let us agree on what we mean by a computer program. A program is a set of instructions made by a human being which a computer follows in order to complete some task and thereby to fulfill some goal of the human programmer. The human being identifies an objective and, by means of the set of instructions in the program, causes the machine to do the work which accomplishes the objective.

Every computer which affects our lives operates according to a program. The digital watch keeps count of the minutes and hours because of a program. A computer calculates the bank account balance by means of a program. Another computer prints the phone bill according to a program.

To understand what a computer program is and what our computer-literate students are involved in, let us take a look at an example of a computer program. Assume that we want our computer to give us the sum of any two numbers that we might wish to give it. The program which the computer would have to

contain would be a series of instructions which the computer would follow in order to do this. It is just such a program that our pocket calculators contain. What would such a program look like--conceptually?

To answer this question, we need to analyze the problem and determine what work needs to be performed. The program would have to accomplish four tasks: it would have to get the first number from us; then it would have to get the second number from us; then it would have to add the two numbers together; then it would have to display the results to us.

Each of these tasks requires the computer to perform certain steps. Each of the first two tasks--getting a number from us--requires the machine to receive the number which is input into it and then to store the number in its memory. Since computers can perform only one operation at a time, these tasks require four steps:

1. Receive number A

2. Store number A in memory location X

3. Receive number B

4. Store number B in memory location Y.

The third task--adding the numbers--is, for a computer, similar to our counting on our fingers. In a special memory location, the computer first places one number and then adds in the second number, just as we, wishing to add 2 and 3, for example, might raise two fingers, then raise three more, and count the result. Thus, the sum of the numbers comes to exist in the special memory location.

The steps necessary to perform this task, then, are:

5. Get number A from memory location X

6. Place number A in special memory location Z

7. Get number B from memory location Y

8. Add number B to number A in memory location Z.

The fourth task--displaying the results--requires two final steps:

9. Get the sum C from special memory location Z

10. Project the sum C on the display unit.

Let us observe what we have done here. We began with a certain intention or goal: to cause the computer to add any two numbers which we give it. To do this, we laid out a logical plan which would accomplish our purpose. This plan contained four tasks, which form a sequence. (See Figure 1, page 27.)

After we had generated the plan, we wrote out the individual steps which are necessary to get the machine to do the job. These steps are the program itself. (See Figure 2, page 27.)

What would remain for us as computer programmers at this point would be, first, to translate the steps of the program into the code that the machine works with; then, to key the program into the machine; and finally, to test-run the program to see that it operates properly. If it does not, then we would study our program, find the errors, fix them, key the corrections into the machine, and re-test.

From the foregoing it is clear that a computer programmer composes for the machine. In many important ways, computer programming and expository composition are identical processes; it is only the audience which differs.

Let us examine these similarities and also, incidentally,

the differences between the processes. We will look first at the products themselves--the program and the essay. Then we will examine the process of creating the products. And finally we will discuss the editing of the finished products.

THE PRODUCTS: THE PROGRAM AND THE ESSAY

We are accustomed to thinking of an essay as an organized series of ideas. You might balk then at the assertion that an essay is a series of instructions which alters the internal state of the reader in a predictable way. The chief offending phrase is probably "series of instructions." It is hard to see ideas as instructions. And indeed they are not. But sentences are. Implicit in every sentence is the instruction to the readers that they must impress the content of the sentence on their intellects. And the way in which their intellects are affected depends in large part on the meaning of the sentence. That is, each sentence causes the contents of the reader's intellect to be re-formed or changed in some way according to the meaning of the sentence. This is at least one of the steps in the process which we call comprehending a reading.

Individual sentences, then, cause certain meanings to be set into the intellect of the reader and thus cause the intellect to be restructured according to those meanings. The

sequence of sentences that constitute the argument of an essay thus cause an arrangement or network of meanings to be established in the intellect. If the individual sentences are well written and express the connections between the meanings that make up the argument--that is, if the instructions on how the reader should form his or her intellect are clear and correct--then, assuming the reader's attention and capacity, a complete intellectual structure will be built in the mind of the reader, and moreover, this structure of meanings will replicate the network of meanings that the writer was attempting to express. That is, the argument as it exists in the reader's mind will match the argument in the writer's mind. Therefore, if the sentence-instructions are clear and correct, the results of the reading will be predictable: the reader will "understand" the argument of the essay.

It is not far-fetched, then, to see an essay as a series of instructions which alter the internal state of the reader in a predictable way. But if this is the case (we humanists will have to swallow hard on this one), then an essay programs the reader in exactly the same way that a computer program programs a computer. The essay and the computer program are identical in abstract form and in function. For, a computer program is a series of instructions which alters the internal state of a computer in a predictable way.

The difference between an essay and a computer program is

essentially the audience. This difference in audience, of course, is quite dramatic. Computers are electronic devices which have no choice but to operate according to the programmed instructions. Readers, on the other hand, are free to accept, to reject, to question, to contest or to forget the meanings which they have allowed to be impressed upon themselves by a reading. In addition, readers, unlike machines, are equipped with a set of emotions which substantiate and give value to the meanings in their intellects.

Moreover, the limited lexical capacity of computers compels the programmer to work in a highly constrained environment with a small inventory of terms and operations, where much of the work is formulaic or repetitious. The vast and varied powers of the human mind, on the other hand, free the writer to work in a highly flexible environment, in which intellect, imagination, and the language combine to offer a staggering array of options at every turn, and in which novelty and inventiveness can pervade not only the content but also often the form of the work.

Nonetheless, what the program does to the computer is in essence what the essay does to the reader. As the censor and the propagandist and the ad-man know about reading, the mind is impressed with the meaning and the reader is changed (or "programmed") by it before any action or reaction on the

reader's part can occur.

THE PROCESSES OF PROGRAMMING AND COMPOSING

Although it may be uncomfortable for humanists or for teachers trained in the traditional composition pedagogy to see an essay as an intellect-programming device, for students experienced in computer programming such a view may seem quite natural. Encouraging this view may prove fruitful, as well. For, once the analogy between the products--the essay and the computer program--has been established, then the process which is used to generate computer programs may function as a striking analogy to the composing process. The students would be able, then, to draw on the concepts derived from their programming experience and to apply these to their understandings of the process of writing.

Let us examine some of these correspondences.

One of the main difficulties in the teaching of composition is convincing students that writing is more than self-expression; it is an act of communication. Since that is so, recognition of the audience--its limitations, its expertise, its prior knowledge, its expectations--is crucial to the production of successful prose. But it is easy for students, or any writer for that matter, to forget the

audience because readers' minds are relatively flexible, they are forgiving of error and obscurity, and they actively seek to understand.

Computer programmers know a different and crueler audience. The machine is completely unforgiving, in the sense that it does not compensate at all for error or ambiguity. It rigidly demands perfection. As a result, programmers, acquainted with the frustration that results from the relentlessness of the machine, are never forgetful of their audience for very long and approach it with respect and care.

Computers demand precision, both in the planning of the program and in its execution. Consequently, the writing of programs is rarely an intuitive or off-the-cuff process. Rather, as we did in our example above, programmers must approach the programming task as a problem to be solved in a systematic way. The first step is to define clearly the objective of the program. Then the problem is broken down into its parts, and the steps which are necessary to solve the problem and reach the objective are identified.

The steps of the program, arrived at in this way, must be explicit, and they must be carefully arranged. It is clear, of course, that every step must be explicitly stated; no step may be left out. It would not do, for example, to ignore Step 4 of our program

4. Store number B in memory location Y

simply because Step 2 is similar. The machine makes no inferences and sees no similarities. If we were to leave out Step 4, then number B would not be stored in memory. Later, the machine, trying to execute Step 7

7. Get number B from memory location Y

would not find number B and the program would fail.

Thus, the machine demands that every step in the sequence of program steps must be clearly articulated. In addition, the sequence in which the steps are placed is crucial to the success of the program.

For example, consider Steps 1-4 in our sample program:

1. Receive number A

2. Store number A in memory location X

3. Receive number B

4. Store number B in memory location Y

The process here is to get a number and store it, then get another number and store it. Now imagine that our programmer has a different idea. Assume that the programmer plans, as seems reasonable, to get both numbers first, and then to store them. The first four steps of the program would then look like this:

*1. Receive number A

*2. Receive number B

*3. Store number A in memory location X

*4. Store number B in memory location Y

The machine will inexorably follow this sequence of instructions. But remember that the machine can perform only one operation at a time. So, it will accept number A from us when we input it. But when we input number B, number A will be lost because the machine cannot hold number A and accept number B at the same time. Thus, on the third instruction it will store in location X the only number it has on hand: number B. Likewise, on instruction 4 it will store number B in location Y. Later, when it adds the two numbers in Step 8, the result will not be the sum of A and B because the number

stored in location X was incorrect.

Thus, the sequence of the steps, or the logic of the program, must be carefully controlled.

The correspondence between these parts of the programming process and composing is obvious. What a good writer does, and what we would like to see our writing students do, is just what the programmer does in fact do.

The establishment of a clear objective is the first priority of both the writer and the programmer. In writing, this objective takes the form of a thesis statement. How to accomplish this objective--for the writer, how to elucidate the thesis statement--is the problem for which the writer or programmer must construct a solution. The solution, of course, is the essay or the program.

Having established an objective, the writer and the programmer each must analyze the problem into its parts and see the relationships between the parts. For the programmer, a flowchart such as in Figure 1 represents these parts and their relationships. The writer's outline, whether formal or merely rough jottings, is the equivalent of this.

Then the individual steps necessary to realize the various parts of the outline or flowchart and to accomplish

the objective must be constructed. In programming, these steps are often first written in English (as we have done in Figure 2) before being coded into a programming language. The purpose for this is to perfect the logic of the program before encoding: to make sure that each step is explicitly stated and in the correct position relative to the other steps. The analog to this process in writing is the production of the sentences and paragraphs of the essay itself.

Moreover, in the cases of both writing and programming, the constraints which the audiences--the reader or the computer--impose on the finished product are the same (although different in degree). Both audiences demand explicit statement of the material. Ambiguity, obscurity, and omission of material are not tolerated.

In addition, both audiences demand coherence. The succession of ideas which a reader experiences must be as logically organized as the succession of commands which the computer obeys. Otherwise, both the essay and the computer program fail.

Thus, because computers demand explicit step-by-step instructions, computer programmers are forced into deliberate planning, clear and coherent thought, and explicit expression. Such valuable habits of mind, it seems to me, are easily transferable to the composing process.

There are other analogies between programming and composing. A computer program may be viewed as a system of parts which solve a problem. From this viewpoint, a program is a set of mutually interacting parts which are organized to produce a certain result. The parts of a program are groups of instructions which accomplish the individual tasks identified in the analysis of the problem. For example, the instructions in our sample program in Figure 2 are divided by spacing into three such parts, called "routines" in computerese: the first routine gets the numbers, the second adds them, and the third displays the sum. These routines interact to accomplish the objective of the program: to produce a sum for any two given numbers. Together the routines form a system.

An essay is similarly a system of parts which solve a problem, when the problem is considered to be the elucidation of the thesis statement. As with the computer program, the parts are the realizations of the major divisions of the outline. The ideas contained in these parts interact with each other to accomplish the objective: the essay taken as a whole produces understanding of the thesis statement.

Viewing both programs and essays as systems is beneficial in two major respects. First, viewed thus, both programs and

essays can be seen to have a cellular structure. Each part, whether it be a routine or a paragraph or a section, is a discreet unit with its own structure and function. But these parts have validity only if they contribute to the accomplishment of the general objective of the program or essay. That is, every part is evaluated in terms of its contribution to larger units of the work.

Thus, each part is a functional unit, and each part should have a clearly explicable function in the structure. The principles of economy and clarity of organization are at issue here. Each part should contribute to the success of the whole in an identifiable way, and there should be no useless parts.

These principles are sometimes not so clear to student writers. Student writing often contains passages which miss the intended point or sometimes are simply padding. On the other hand, to programmers these principles are fundamental. Computer code is not nearly so easy to produce as are words, and no one who writes a program wants to do more work than is necessary. And so, the operative question becomes, "What **must** be done to get from point A of the plan to point B?"

Programmers, then, by virtue of their experience, recognize the value of organizational clarity and of economy.

They may thus be equipped to understand that essays, like programs, are systems of functional cells, each with its own structure and with an explicable function.

Furthermore, if programs and essays are systems, then like all systems, they have two general aims: first, to function smoothly, and second, to protect themselves from distress. To a computer program, distress comes in the form of the unanticipated. Programs are designed to process only the data or information which the programmer anticipates that they will process. If unanticipated data are injected into a program, it will fail.

For example, the first four steps of our sample program assume that the input into the computer is in the form of arabic numbers: 0, 1, 2, ... Consider what would happen if we were to input, accidentally or intentionally, a letter or a Roman numeral into this program. In such a case, the addition in Step 8 could not take place, and so the program would be unable to fulfill its objective, which is to provide a sum. In short, the program would fail.

The programmer must anticipate such contingencies, in order to protect the program from failure and to assure its smooth operation. One way to do that in our sample program would be to insert appropriate lines into the first routine as

follows:

1. Receive number A

1a. If A is not a number, repeat Step 1; otherwise, perform Step 2

2. Store number A in memory location X

3. Receive number B

3a. If B is not a number, repeat Step 3; otherwise, perform Step 4

4. Store number B in memory location Y.

The programmer, then, must develop the foresight to anticipate distress to the operation of the system.

In the realm of expository writing, system distress occurs in the form of objections to the arguments or the evidence offered in the essay. Objections challenge the credibility of the structure of ideas which constitutes the essay, and thus they threaten the achievement of the essay's objective of reader understanding. In other words, objections

threaten the smooth functioning of the essay as a system.

Our writing students are taught, of course, to anticipate and answer objections to their essays. But this is not easy for them to do, for it demands that they have an appreciation for the undesirable.

Programmers, however, as we have just seen, have experience in just this sort of anticipating. Such experience should provide a foundation for gaining control of this somewhat sophisticated skill.

It is profitable, then, for writers to see the essay as a system, an interaction of its parts to produce a tight, economical argument. Such a view allows writers to evaluate each sub-part of the essay in terms of its contribution to the larger argument. It also allows them to identify and discard sub-parts which have no clear function. And finally, it allows them to anticipate the objections and counter-arguments which pose hazards to the success of their essays. To programmers who are studying composition, such a view of essays as systems may be quite natural.

EDITING

One final issue deserves attention from our present viewpoint: the editing process. Of all the parts of the composing process, editing is the one most disdained and most likely to be ignored or treated casually by students. The reason for this, of course, is that the rewards of precise editing are received only after much effort and are too far removed from whatever pleasure the students may feel in expressing their ideas. To many students, editing is simply labor to satisfy the teacher; it has no real value.

In addition, editing means facing one's mistakes, often over and over. It is a frustrating activity that sometimes causes embarrassment or self-belittlement. And often teachers, as well as the student writers themselves, take part in the incrimination ritual that can accompany editing. It is no wonder that writing students often hold editing in slight regard.

There are two reasons why the art of editing is put in this sorry state. The first is that writers who lack a sufficient understanding of their audience do not comprehend the damage that errors in organization, logic and mechanics cause in the reading process. They do not see how such errors victimize their readers. It is when students lack this appreciation of their readers' troubles that errors in their

writing cause no concern.

The second reason is a consequence of the first. When writers lack a vision of their audience, then errors are not seen as interruptions of the communication process; rather, they are seen as sources of blame. Attention is turned from the writing to the writer. Errors are seen not as problems within the communication system; they are seen as faults of the writer. By personalizing errors in this way, students discredit themselves as writers.

None of this pain and self-reproach is necessary. The whole problem arises because in writing as in many intellectual activities, feedback on the success of the work is often impossible within a time span in which it would be helpful. In most circumstances, when writers make errors in their work, the reactions which the errors cause in the reader occur in situations which are remote in time or space. For example, the teacher reads the student's essay at home and returns it a week later--long after the student's concentration on it has dissolved. Likewise, the journal editor reads the teacher's article six months after submission and in another city, and communicates with the writer by cryptic postcards. The writer is isolated from reader reactions. Because the conventions of writing provide no immediate feedback mechanism, the writer has difficulty associating the errors with the functioning essay itself;

instead, the writer turns blame for the errors on himself.

The opposite situation is true of programming. As the mathematician, Seymour Papert, points out about children learning computing, feedback on the success or failure of a program is so immediate that children-programmers have no difficulty in associating errors with the program rather than with faults in themselves.[2] The computer responds instantaneously: either the program runs or it doesn't. When it doesn't, the next move is clear. No self-reproach is necessary, nor will it do any good. What is necessary is to turn back to the program, find the flaw, and fix it. This is clearly a healthy and productive attitude.

What is true for Papert's children-programmers is true for all programmers. One of the attractive aspects of computing is the speed of the machine's response to one's work. Within seconds, the programmer knows the result: the program either runs or it doesn't. If it doesn't, there is nothing useful to do except to find the flaws and to fix them.

Students experienced in programming, then, bring to their intellectual activity a healthy and productive attitude toward error in their work. If these students can be shown that an essay, like a program, is a system, then with regard to editing out their errors, as in the other aspects of planning and execution of the work discussed above, the students have

at their disposal a set of attitudes and concepts which may guide them to a useful understanding of the composing process.

AN EPILOGUE

When I first conceived these ideas, I thought them to make a strong case for the assertion that computer literacy should be taught not by technicians or computer professionals but by humanists who are appropriately competent with computers. I thought that humanists, by virtue of their experience with the arts, and by virtue of the philosophical bent and the global concerns of their training, would be able to develop in students not only the necessary skills and attitudes for programming, but also an understanding of the computer as a model for other kinds of human intellectual endeavor. Since the computer is now so fashionable and students are eager to be involved with it, it seemed to me that out of such enthusiasm a computer literacy instructor who is a humanist could fashion in students attitudes and concepts which are applicable to a broad range of knowledge.

I still believe that this is the best approach to the teaching of computer literacy. But I have come since then to see that even if the debate over who should teach computer literacy is won by the technicians, the hope of extending the computer experience into other realms is not lost. What is

then required is that instructors in the individual subject areas be knowledgeable about the world of computing, not only--or even primarily--to use the software available to them as teaching aids, but so that they may translate their students' proudly held experience in computing into the attitudes and concepts needed to explore other subject areas.

Papert, in his seminal book, Mindstorms, explores how this can be done with certain concepts in mathematics. I have tried in this paper to show how this could be done in the area of English composition. I recognize of course that there are many ways in which computer programming and composing expository prose are far from identical processes. Nonetheless, many of the fundamental concepts of both crafts are analogous, in terms both of the crafted product and of the processes used to craft them. Confronted with students who have programming experience, writing teachers have a place to start, a foundation on which to build, if they make use of these correspondences.

The teacher's central problem is to lead students to experience the concepts which are being taught. Once a student seriously undergoes the experience of analyzing a problem into its parts, or of worrying out the structure of an argument, or of identifying a thesis statement, or of recognizing an ambiguity, then to that extent that student has become a real writer and his or her eyes are opened to the

craft and its problems. Until the student has such experiences, he or she is simply imitating.

When a writing instructor faces students experienced in programming, those students have already experienced in the realm of programming many of the concepts and attitudes which the instructor hopes to instill in them with respect to composing. The instructor's task in that case is to bring those students to recognize the correspondences between what they have already experienced and what they must now learn.

NOTES

1 See, for example, Helen J. Schwartz and Lillian S. Bridwell, "A Selected Bibliography on Computers in Composition," College Composition and Communication, 35 (1984), 71-77; James E. Davis and Hazel K. Davis, eds., Computers in Teaching English, Vol. 9 of FOCUS: Teaching English Language Arts (Ohio University at Athens) Spring, 1983.

2 Seymour Papert, Mindstorms: Children, Computers, and Powerful Ideas (New York: Basic Books, 1980),23,114.

Figure 1

Logical Plan (Flowchart)

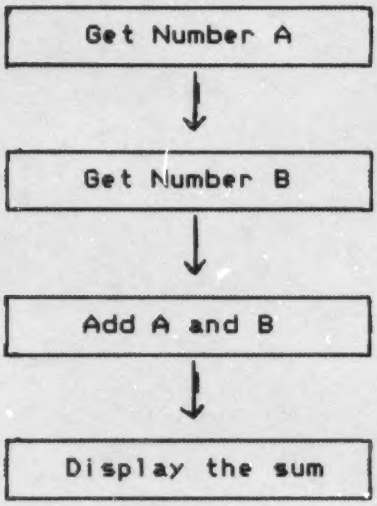

Figure 2

The Steps of the Program

1. Receive number A
2. Store number A in memory location X
3. Receive number B
4. Store number B in memory location Y
5. Get number A from memory location X
6. Place number A in special memory location Z
7. Get number B from memory location Y
8. Add number B to number A in memory location Z
9. Get the sum C from special memory location Z
10. Project the sum C on the display unit.